Available online at www.sciencedirect.com

SCIENCE DIRECT®

Discrete Optimization 2 (2005) 154–167

DISCRETE OPTIMIZATION

www.elsevier.com/locate/disopt

The two-dimensional bin packing problem with variable bin sizes and costs

David Pisinger*, Mikkel Sigurd

Department of Computer Science, University of Copenhagen, Universitetsparken 1, DK-2100 Copenhagen, Denmark

Received 24 June 2004; received in revised form 8 January 2005; accepted 14 January 2005

Abstract

The two-dimensional variable sized bin packing problem (2DVSBPP) is the problem of packing a set of rectangular items into a set of rectangular bins. The bins have different sizes and different costs, and the objective is to minimize the overall cost of bins used for packing the rectangles. We present an integer-linear formulation of the 2DVSBPP and introduce several lower bounds for the problem. By using Dantzig–Wolfe decomposition we are able to obtain lower bounds of very good quality. The LP-relaxation of the decomposed problem is solved through delayed column generation, and an exact algorithm based on branch-and-price is developed. The paper is concluded with a computational study, comparing the tightness of the various lower bounds, as well as the performance of the exact algorithm for instances with up to 100 items.

Keywords: Bin-packing problem; Column generation; Constraint programming; Cutting and packing

1. Introduction

During the last four decades the bin packing problem (BPP) has received a great deal of attention and it is today one of the most well-studied problems within the field of combinatorial optimization. The reason for this popularity is partly due to the simplicity of the formulation and partly due to the wide number of applications. In its simplest form, the BPP may be formulated as: given n items with sizes $w_i \in [0, 1]$, $i = \{1, \ldots, n\}$, and an infinite number of bins of size 1, find the smallest number of bins needed to pack all items.

The problem may not seem very practical at first glance, but surprisingly many problems can either be reduced to the BPP or contains the BPP as a subproblem. This includes applications within the fields of packing, cutting, scheduling and communications.

The problem has been solved with a variety of algorithms. Especially, a huge amount of work has been done on on-line and off-line approximation algorithms. See Coffmann et al. [10] for a survey on approximation algorithms and Csirik and Woeginger [11] for an overview of on-line algorithms. Also, a number of exact algorithms have been developed, using the branch-and-bound technique where lower bounds have been derived either using geometric arguments or using Dantzig–Wolfe decomposition [24].

The two-dimensional bin packing (2DBPP) has been intensively studied in the literature. See e.g. Lodi et al. [22] for a survey and Belov [3] for a comprehensive overview of models and algorithms. Exact algorithms and various lower bounds for the 2DBPP have been presented in [6,7,12,13,25]. The latter two papers allow the rectangles to be rotated by 90 degrees. Higher-dimensional generalizations have also received some attention (e.g. [9,23]).

* Corresponding author. Tel.: +45 35 32 13 54; fax: +45 35 32 14 01.
E-mail addresses: pisinger@diku.dk (D. Pisinger), sigurd@diku.dk (M. Sigurd).

1572-5286/$ - see front matter
doi:10.1016/j.disopt.2005.01.002

The basic BPP only allows bins of the same size. A natural generalization is the *variable-sized BPP* (VSBPP) where various bin sizes are allowed. This problem is also known as the *cutting stock problem with multiple stock sizes*, considered in [4,17]. Friesen and Langston [15] presents three approximation algorithms for the 1DVSBPP and Monaci [26] presents lower bounds and an exact algorithm.

In this paper we address the exact solution of the 2DVSBPP. The bin types may have *variable costs*, i.e. costs which are not proportional to the size of the bin, and the objective is to minimize the cost of the bins used. Such problems have several applications in industry, e.g. in cutting of wood where small pieces of wood may be relatively cheaper than larger pieces or in cutting of metal plates, where some standard-sized plates may be relatively cheaper than non-standard-sized plates. A more formal definition of the variable cost 2DVSBPP is presented in Section 2.

Hopper and Turton [18] consider a variant of the 2DVSBPP where there is a bounded number of each bin type, and uniform bin costs are used. The 1DVSBPP where bin costs equal the bin sizes is considered in [26], while Chen et al. [9] consider the 3D version of the same problem.

In Section 3, we present several lower bounds for the 2DVSBPP, and evaluate their performance. The performance of a lower bound algorithm L is given by the worst-case performance ratio

$$R_L = \inf_{I \in P} \left\{ \frac{L(I)}{OPT(I)} \right\}, \tag{1}$$

where P is the set of all instances, $L(I)$ is the objective value returned by algorithm L on instance I, and $OPT(I)$ is the optimal objective value for I. We generalize two lower bounds originating from ordinary 1D and 2D bin packing in Section 3 and introduce a new lower bound based on integer programming in Section 3.1. In Section 3.2, we present a lower bound based on Dantzig–Wolfe decomposition of the 2DVSBPP. This lower bound is used in a branch-and-price algorithm which solves the 2DVSBPP to optimality. The branch-and-price algorithm is adapted from the algorithm recently proposed by Pisinger and Sigurd [30] for solving the normal 2DBPP. The algorithm is briefly outlined in Section 4.

We conclude the paper in Section 5 by presenting a set of new benchmark tests for the 2DVSBPP with variable costs along with our computational results. The experimental quality of the lower bounds presented is reported along with running times for our exact algorithm.

2. Problem formulation

Assume that a set $\mathscr{R} = \{1, \ldots, n\}$ of rectangular *items* are given, item i having *width* w_i and *height* h_i. Moreover, m rectangular *bin types* are available, bin type $k = 1, \ldots, m$ having *width* W_k, *height* H_k and *cost* c_k. It is assumed that an infinite amount of each bin type is available.

The 2DVSBPP with variable costs, asks to pack all the items into appropriate bins so that no two rectangles intersect and so that the overall cost of the bins used is minimized. In the integer-valued version of the 2DVSBPP we assume that all coefficients are positive integers. Only non-negativity constraints are present in the real-valued version.

We assume that all items fit into at least one bin type, as otherwise no solution to the problem exists. If no items fit into a bin type, then the given bin type may be deleted from the problem. Finally, one may remove dominated bin types. If bin types k_1 and k_2 satisfy

$$c_{k_2} \geqslant c_{k_1} \quad \text{and} \quad W_{k_2} \leqslant W_{k_1} \quad \text{and} \quad H_{k_2} \leqslant H_{k_1}$$

we may remove bin type k_2, since we can always find an optimal solution which does not make use of the dominated bin type k_2.

Although we assume that an *unbounded* amount of bins is available of each bin type, it is obvious that we cannot use more than n bins of any type. Hence, we can transform the problem to a *binary* version where we have $m' := mn$ separate bins, and each bin can only be used once. As the two formulations of 2DVSBPP are equivalent, we will use the binary version where appropriate, shortly stating that we are looking at the binary model. Note that in the binary model it is easy to handle upper bounds on the availability of each bin, since we simply only introduce as many copies of each bin type as stated by the corresponding upper bound.

To formulate the variable cost 2DVSBPP in *binary* form as an IP model we use the modeling technique proposed by Onodera et al. [28] and Chen et al. [9]. The following decision variables are used: the binary variable ℓ_{ij} is 1 iff item i is located left to j, and similarly b_{ij} is 1 iff item i is located below j. The binary variable f_{ik} attains the value 1 iff item i is located in bin k. The variable z_k is 1 iff bin k is used. Finally, (x_i, y_i) are the lower left coordinates of item i.

To ensure that no two items overlap, the following inequality must hold:

$$\ell_{ij} + \ell_{ji} + b_{ij} + b_{ji} + (1 - f_{ik}) + (1 - f_{jk}) \geqslant 1, \quad i, j \in \mathscr{R},\ i < j,\ k \in \{1, \ldots, m'\}. \tag{2}$$

If i is located left to j, i.e. $\ell_{ij} = 1$ then we have that $x_i + w_i \leqslant x_j$. In general, we have the constraints

$$\begin{aligned} &\ell_{ij} = 1 \Rightarrow x_i + w_i \leqslant x_j, \\ &b_{ij} = 1 \Rightarrow y_i + h_i \leqslant y_j. \end{aligned} \tag{3}$$

No part of the items may exceed the bin, hence if i is placed in bin k then $0 \leqslant x_i \leqslant W_k - w_i$ and $0 \leqslant y_i \leqslant H_k - h_i$. Moreover, every item i has to be placed in some bin so $\sum_{k=1}^{m'} f_{ik} \geqslant 1$. Bin k is used if at least one item is placed in k so $\sum_{i \in \mathscr{R}} f_{ik} > 0 \Rightarrow z_k = 1$.

Let $W = \max_{k \in \{1,\ldots,m'\}}\{W_k\}$ and $H = \max_{k \in \{1,\ldots,m'\}}\{H_k\}$, be the maximum bin width and bin height. Using standard modeling techniques for IP models [32], the 2DVSBPP problem in *binary* form can now be formulated as the following MIP problem

$$\begin{array}{lll} \text{minimum} & \displaystyle\sum_{k=1}^{m'} c_k z_k & \\ \text{subject to} & \ell_{ij} + \ell_{ji} + b_{ij} + b_{ji} + (1 - f_{ik}) + (1 - f_{jk}) \geqslant 1, & i, j \in \mathscr{R},\ i < j,\ k \in \{1, \ldots, m'\}, \\ & x_i - x_j + W\ell_{ij} \leqslant W - w_i, & i, j \in \mathscr{R}, \\ & y_i - y_j + Hb_{ij} \leqslant H - h_i, & i, j \in \mathscr{R}, \\ & x_i \leqslant W_k - w_i + (1 - f_{ik})W, & i \in \mathscr{R},\ k \in \{1, \ldots, m'\}, \\ & y_i \leqslant H_k - h_i + (1 - f_{ik})H, & i \in \mathscr{R},\ k \in \{1, \ldots, m'\}, \\ & \displaystyle\sum_{k=1}^{m'} f_{ik} \geqslant 1, & i \in \mathscr{R}, \\ & f_{ik} \leqslant z_k, & i \in \mathscr{R},\ k \in \{1, \ldots, m'\}, \\ & \ell_{ij}, b_{ij} \in \{0, 1\}, & i, j \in \mathscr{R}, \\ & f_{ik} \in \{0, 1\}, & i \in \mathscr{R},\ k \in \{1, \ldots, m'\}, \\ & x_i, y_i \geqslant 0, & i \in \mathscr{R}, \\ & z_k \in \{0, 1\}, & k \in \{1, \ldots, m'\}. \end{array} \tag{4}$$

The model contains $O(mn^2)$ binary variables, $O(n)$ continuous variables and $O(mn^3)$ constraints. Computational studies have shown that the model is very difficult to solve by standard MIP-solvers. This motivates the development of specialized algorithms, in particular searching for tight lower bounds that can be used in various enumerative algorithms.

3. Lower bounds

In this section we present several lower bounds for the 2DVSBPP. We will make use of a cost-dependent ceiling function $\lceil \cdot \rceil_c$.

$$\lceil a \rceil_c := \min\{c_k x_k \mid c_k x_k \geqslant a,\ x_k \in \{0, 1\},\ k = 1, \ldots, m'\}. \tag{5}$$

The problem (5) can be recognized as a subset-sum problem in minimization form which can be solved in pseudo-polynomial time if all coefficients are integers. The ceiling function allows us to round up lower bounds to the nearest achievable cost-sum. If the number of distinct solutions to (5) as function of a is moderate, one may precompute all solutions through dynamic programming and store them in a table. This means that a branch-and-bound function can compute the ceiling by a table lookup.

Monaci [26] presents several lower bounds for the special case of the 1DVSBPP where costs equal bin sizes. Furthermore, several lower bounds for the normal 2DBPP have been presented during the last decade. Martello and Vigo [25] derived lower bounds for the 2DBPP based on geometric properties of bins and items, while Fekete and Schepers [13] presented lower bounds for the 2DBPP based on *dual feasible functions* and a graph theoretic representation of bin packings.

Neither the lower bounds for the 1DVSBPP nor the lower bounds for the 2DBPP are valid for the 2DVSBPP. The techniques used by Martello and Vigo and by Fekete and Schepers for the 2DBPP are heavily dependent on the fact that all bins have the same size. Thus, it is not possible to generalize these lower bounds for multiple bin sizes.

We can, however, adapt the lower bounds from the 1DVSBPP by relaxing some of the constraints in the model for the 2DVSBPP. In the 1DVSBPP we are given n items with sizes w_i, $i = 1, \ldots, n$, and m bin types with sizes W_k and costs c_k,

$k = 1, \ldots, m$. The 1DVSBPP with arbitrary costs can be formulated in *binary* form by the following model:

$$\begin{array}{lll} \text{minimize} & \sum_{k=1}^{m'} c_k y_k & \\ \text{subject to} & \sum_{k=1}^{m'} x_{ik} = 1, & i = 1, \ldots, n, \\ & \sum_{i=1}^{n} w_i x_{ik} \leqslant W_k y_k, & k = 1, \ldots, m', \\ & x_{ik} \in \{0, 1\}, & i = 1, \ldots, n,\ k = 1, \ldots, m' \\ & y_k \in \{0, 1\}, & k = 1, \ldots, m', \end{array} \tag{6}$$

where $m' := mn$ is an upper bound on the total number of bins used, x_{ik} is a binary variable taking the value 1 if item i is placed in bin k, and y_k is a binary variable taking the value 1 if bin k is used.

Monaci [26] presents several lower bounds for the special case of the 1DVSBPP where the bin costs equal the bin sizes. These lower bounds can easily be extended to the general case where arbitrary bin costs are allowed. Since all items have to be placed in some bin, our solution must contain a bin that can hold the biggest item, i.e. a valid lower bound is

$$L_w := \left\lceil \min_{k=1,\ldots,m} \{c_k : W_k \geqslant w_i,\ i = 1, \ldots, n\} \right\rceil_c .$$

In the 2D case we do not have a complete ordering of the bin sizes. Hence, we cannot determine a cheapest bin that has room for all items—in fact such a bin may not exist. Instead we look at the sets B_i, $i = 1, \ldots, n$, which are the set of bin types that can hold item i, i.e.

$$B_i := \{k \in \{1, \ldots, m\} : W_k \geqslant w_i \wedge H_k \geqslant h_i\}. \tag{7}$$

Any feasible solution must contain at least one bin from each of the sets B_i. Let $c_{k_i} := \min_{k \in B_i} c_k$ be the cost of the cheapest bin in B_i. Then

$$L_{2w} := \left\lceil \max_{i=1,\ldots,n} c_{k_i} \right\rceil_c$$

is a lower bound on the solution value, and it can be evaluated in polynomial time $O(nm)$. The worst-case performance ratio (1) of L_{2w} is $1/n$. The performance ratio is achieved using the following instance I_0: we have n items of size 1×1, one bin type of size 1×1 and cost 1. For this instance $L_{2w}(I_0) = 1$ and $OPT(I_0) = n$. On the other hand, the L_{2w} lower bound is never worse than $1/n$ since the bin chosen in L_{2w} is the most expensive bin type needed in the solution. Thus, we can construct a feasible packing using n bins, which are all cheaper than L_{2w}, yielding a total cost of at most $n \cdot L_{2w}$.

As demonstrated by the worst-case instance I_0, the problem with the L_{2w} lower bound is that it only depends on *the* most expensive item to pack. The *continuous lower bound* L_c accommodates this problem. In the 1D case with bin costs equal to bin sizes the bound $L_c^e = \sum_{i=1}^{n} w_i$ is obviously a lower bound on the optimal value. For the 2D case we consider the item areas, thus if bin costs equal bin sizes, then $L_{2c}^e = \sum_{i=1}^{n} w_i h_i$ is a lower bound on the optimal value. For the general case with arbitrary bin costs the continuous lower bound is

$$L_{2c} := \left\lceil \frac{c_{k_0}}{W_{k_0} H_{k_0}} \sum_{i=1}^{n} w_i h_i \right\rceil_c ,$$

where k_0 is the bin type with cheapest cost/area ratio. For the instance I_0 proving the worst-case performance of bound L_{2w}, the continuous lower bound equals the optimal solution value. However, L_{2c} can perform arbitrarily bad, as the following instance I_1 shows: we have two items of dimensions $(w_1, h_1) = (1, 2)$ and $(w_2, h_2) = (2, 1)$. Moreover, we have two bin types of the same dimensions $(W_1, H_1) = (1, 2)$ and $(W_2, H_2) = (2, 1)$, having associated costs $c_1 = \varepsilon$ and $c_2 = 1$. Since bin type $k = 1$ has the cheapest cost/area ratio we get $L_{2c}(I_1) = 2\varepsilon$. The optimal value for I_1 is $OPT(I_1) = 1 + \varepsilon$, hence the worst-case performance ratio is $R_{L_{2c}} = 2\varepsilon/(1 + \varepsilon)$ which can be made arbitrarily small independent of n and m.

The worst-case example exploits the fact that L_{2c} bounds by the cheapest bin cost. However, possibly not all items fit into the cheapest bin. We can achieve a better bound by modifying the continuous lower bound, so that the cost of packing each item is a fraction of the cost of a bin in which the item fits. Let $l_i \in B_i$ be the bin with the cheapest cost/area ratio $c_{l_i}/(W_{l_i} H_{l_i})$ in which item i fits. The modified continuous lower bound is thus

$$L'_{2c} := \sum_{i=1}^{n} \frac{w_i h_i c_{l_i}}{W_{l_i} H_{l_i}}, \tag{8}$$

where we may tighten the lower bound to $\lceil L'_{2c} \rceil_c$. The bound may be evaluated in O(nm). The L'_{2c} lower bound equals the optimal solution value on both of the above instances I_0 and I_1. But the worst-case performance ratio of the L'_{2c} lower bound is still arbitrarily bad, as the following instance I_2 shows: we are given an item of size 1×1, an item of size 1×2, a bin type of size 1×1 with cost 1 and a bin type of size $M \times M$ with cost M. On this instance

$$\lceil L'_{2c}(I_2) \rceil_c = \left\lceil \frac{M}{M^2} + \frac{2M}{M^2} \right\rceil_c = 1$$

but the optimal value is M, which means that the worst-case performance ratio is $R_{L'_{2c}} = 1/M$ which can be arbitrarily small independent of n and m.

Naturally, the maximum of L_{2w} and L'_{2c} is also a valid lower bound,

$$L_{2wc} := \lceil \max \{L_{2w}, L'_{2c}\} \rceil_c = \left\lceil \max \left\{ \sum_{i=1}^{n} \frac{w_i h_i c_{l_i}}{W_{l_i} H_{l_i}},\ c_{k_1}, \ldots, c_{k_n} \right\} \right\rceil_c,$$

which yields the optimal solution for the above two instances. Since the L_{2w} bound has a worst-case performance of $1/n$, the L_{2wc} will have a worst-case performance which is no worse than $1/n$. The following instance I_3 yields a performance of $1/n$, thus we can conclude that the bound is tight: we are given two bin types of sizes $(W_1, H_1) = (10n, 10n)$ and $(W_2, H_2) = (1, 1)$ and with costs $c_1 = 10n$, $c_2 = 1$. We have n items of size 1×1. On this instance

$$L_{2wc} = \max \left\{ 1, \sum_{i=1}^{n} \frac{10n}{100n^2} 1 \right\} = \max \left\{ 1, \frac{1}{10} \right\} = 1,$$

while the optimal solution value is n.

However, for the following restricted class of problem instances, L_{2wc} has a worst-case performance ratio of $\frac{1}{4}$.

Proposition 1. *Consider the restricted class of 2DVSBPP instances in which the cost/area ratios are the same for all bin types and in which the bins sizes are totally ordered, i.e.* $W_1 \leqslant W_2 \leqslant \cdots \leqslant W_n$ *and* $H_1 \leqslant H_2 \leqslant \cdots \leqslant H_n$. *The* L_{2wc} *lower bound has a worst-case performance ratio of* $\frac{1}{4}$ *on this class of instances.*

Proof. Assume $L_{2w} \geqslant L'_{2c}$: since we have a total ordering of the bin sizes, there must be a smallest bin q where all items fit. Bin q will also be the cheapest bin where all items fit since the cost/area ratio is the same for all bins. According to our assumption $L'_{2c} \leqslant L_{2w} = c_q$ which means that the sum of the item areas is less or equal to the area of bin q. Theorem 1 from Martello and Vigo [25] tells us that we can pack all items in 4 bins of type q.

Now, assume $L_{2w} < L'_{2c}$: let bin q be the smallest (and cheapest) bin in which all items fit. According to Theorem 1 from Martello and Vigo [25] we can pack all items using no more that $4L'_{2c}$ bins of type q. □

3.1. An IP area cover lower bound

The continuous lower bound L'_{2c} can be formulated as a linear program as follows: let B_i be the set of bins that can hold item i as defined in (7). For each item i and bin type $k \in B_i$ we are given a decision variable x_k^i which tells us how big a fraction of bin type k is used to cover item i. The following linear program is equivalent to the L'_{2c} lower bound

$$\begin{aligned} L'_{2c} \quad := \quad & \text{minimize} \quad \sum_{i=1}^{n} \sum_{k \in B_i} c_k x_k^i \\ & \text{subject to} \quad \sum_{k \in B_i} W_k H_k x_k^i \geqslant w_i h_i, \quad i = 1, \ldots, n, \\ & \qquad\qquad 0 \leqslant x_k^i \leqslant 1, \qquad\qquad i = 1, \ldots, n,\ k \in B_i. \end{aligned} \tag{9}$$

If there is a unique bin type $k_i \in B_i$ with lowest cost/area ratio for each item, only $x_{k_i}^i$ will be positive at value $x_{k_i} = (w_i h_i)/(W_{k_i} H_{k_i})$ for each item i in the optimal solution.

In the solution to the 2DVSBPP it is not allowed to use bins fractionally. Thus, we can tighten model (9) by demanding that each bin type is used an integer number of times. We do this by introducing a variable y_k for each bin type $k = 1, \ldots, m$. The y_k variable equals the sum of bins of type k which is used in the solution to (9), and we demand that the y_k variables are integer.

A tighter lower bound L_{IP} is thus given by the following *mixed integer program*:

$$\begin{aligned} L_{IP} := \text{minimize} \quad & \sum_{i=1}^{n} \sum_{k\in B_i} c_k x_k^i \\ \text{subject to} \quad & \sum_{k\in B_i} W_k H_k x_k^i \geqslant w_i h_i, \quad i = 1,\dots,n, \\ & y_k - \sum_{\{i\mid k\in B_i\}} x_k^i = 0, \quad k = 1,\dots,m, \\ & 0 \leqslant x_k^i \leqslant 1, \quad i = 1,\dots,n,\ k \in B_i, \\ & y_k \in \mathbb{N}_0, \quad k = 1,\dots,m. \end{aligned} \tag{10}$$

Since L'_{2c} is a relaxation of L_{IP}, obviously $L'_{2c} \leqslant L_{IP}$. As we shall see in the results section, the inequality is strict for most of the considered problem instances, although the improvement is not large: it is well-known that imposing integrality on a variable larger than one does not tighten the bound very much. Since (10) is a mixed integer program, finding L_{IP} is $\mathcal{NP}$-hard. In practice (10) can easily be solved with standard integer programming solvers like CPLEX [19], since the model is quite small and the number of integer variables is moderate.

3.2. Dantzig–Wolfe decomposition

Gilmore and Gomory [16] showed how Dantzig–Wolfe decomposition may be used for the ordinary BPP. The technique is easily generalized to 2DVSBPP. Assume that $\mathcal{P}_k$ is the set of all feasible packings of a single bin of type $k = 1,\dots,m$ and let $\mathcal{P} = \bigcup_{k=1}^{m} \mathcal{P}_k$ be the set of all feasible packings. For every feasible packing $p_k \in \mathcal{P}_k$, we use the binary decision variable x_{p_k} to model whether packing p_k is chosen in the solution of the decomposed model. Let the constants $\delta_i^{p_k}$ equal 1 iff packing $p_k \in \mathcal{P}$ contains item $i \in \mathcal{R}$ and 0 otherwise. Using the above variables the decomposed formulation of the 2DVSBPP is

$$\text{minimize} \quad \sum_{k=1}^{m} \sum_{p_k\in\mathcal{P}_k} c_k x_{p_k} \tag{11a}$$

$$\text{subject to} \quad \sum_{k=1}^{m} \sum_{p_k\in\mathcal{P}_k} \delta_i^{p_k} x_{p_k} \geqslant 1, \quad i \in \mathcal{R}, \tag{11b}$$

$$x_{p_k} \in \{0,1\}, \quad k = 1,\dots,m,\ p_k \in \mathcal{P}_k. \tag{11c}$$

The objective function minimizes the total cost of the bins used in the solution while constraints (11b) state that every item must be included in some bin.

The LP-relaxation of the decomposed model yields a lower bound L_{DW} for the 2DVSBPP

$$L_{DW} = \text{minimize} \quad \sum_{k=1}^{m} \sum_{p_k\in\mathcal{P}_k} c_k x_{p_k} \tag{12a}$$

$$\text{subject to} \quad \sum_{k=1}^{m} \sum_{p_k\in\mathcal{P}_k} \delta_i^{p_k} x_{p_k} \geqslant 1, \quad i \in \mathcal{R}, \tag{12b}$$

$$0 \leqslant x_{p_k} \leqslant 1, \quad k = 1,\dots,m,\ p_k \in \mathcal{P}_k, \tag{12c}$$

where we may tighten the lower bound to $\lceil L_{DW} \rceil_c$. In the case of the 2DBPP with equal bin sizes, the LP-relaxation gives a fairly tight lower bound on the IP solution as shown by Pisinger and Sigurd [30]. Our results show that this is also true for the variable sized 2DBPP.

However, the decomposed model contains a variable for every feasible packing of a single bin. In general there exists an exponential number of feasible single bin packings, and even for a small number of items, model (12) would be too big to solve. Fortunately, by using *delayed column generation* [16] we can solve the linear program without considering a majority of the variables explicitly. In Section 5 we show, that this lower bound performs very well in practice. We use the lower bound in a branch-and-bound algorithm for solving the 2DVSBPP to optimality. For a recent survey of delayed column generation combined with branch and bound, see [1].

4. A branch-and-price algorithm

In Pisinger and Sigurd [30] we showed how the 2DBPP can be solved by a branch-and-price algorithm using constraint programming to solve the pricing problem. It turns out, that this algorithm can be generalized to handle the 2DVSBPP. We refer to Pisinger and Sigurd [30] for a detailed description of the branch-and-price algorithm to solve the 2DBPP. In the present paper, we only describe the algorithm briefly and we emphasize on the differences between the 2DBPP algorithm and the generalized 2DVSBPP algorithm.

4.1. Delayed column generation

In delayed column generation we solve the linear problem (12) for a subset $\mathscr{P}'$ of the feasible packings $\mathscr{P}$, gradually adding new packings $p \in \mathscr{P} \setminus \mathscr{P}'$ based on the Dantzig rule for solving the linear problem. The *restricted master problem* (RMP) is

$$\text{minimize} \quad \sum_{k=1}^{m} \sum_{p_k \in \mathscr{P}'_k} c_k x_{p_k} \tag{13a}$$

$$\text{subject to} \quad \sum_{k=1}^{m} \sum_{p_k \in \mathscr{P}'_k} \delta_i^{p_k} x_{p_k} \geqslant 1, \quad i \in \mathscr{R}, \tag{13b}$$

$$0 \leqslant x_{p_k} \leqslant 1, \quad k = 1, \ldots, m, \ p_k \in \mathscr{P}'_k. \tag{13c}$$

Here $\mathscr{P}' = \bigcup_{k=1}^{m} \mathscr{P}'_k \subseteq \mathscr{P}$ is chosen such that a feasible solution exists for RMP. Initially we let $\mathscr{P}'$ be the packings of a feasible solution found by a heuristic algorithm. Since we do not remove columns from the RMP in the following iterations, the linear program will always have a feasible solution.

In every iteration of the column generation we solve the RMP. Let π_i, $i \in \mathscr{R}$, be the dual variables of the optimal dual solution to the RMP. The dual linear program of (12) is given by

$$\text{maximize} \quad \sum_{i \in \mathscr{R}} \pi_i \tag{14a}$$

$$\text{subject to} \quad \sum_{i \in \mathscr{R}} \pi_i \delta_i^{p_k} \leqslant c_k, \quad k = 1, \ldots, m, \ p_k \in \mathscr{P}_k, \tag{14b}$$

$$\pi_i \geqslant 0, \quad i \in \mathscr{R}. \tag{14c}$$

The *reduced cost* of packing p_k with respect to the dual variables π_i is

$$c_{p_k}^{\pi} = c_k - \sum_{i \in \mathscr{R}} \delta_i^{p_k} \pi_i.$$

In every iteration of the column generation procedure we find a feasible packing $p \in \mathscr{P}$ with smallest reduced cost c_p^{π}. If $c_p^{\pi} \geqslant 0$ the LP problem corresponding to (11) has been solved to optimality. Otherwise we add the packing p to the RMP and re-optimize. Finding a packing with smallest reduced cost is known as the pricing problem of the column generation procedure. We use the constraint programming algorithm and the heuristic filling algorithms described in Pisinger and Sigurd [30] to solve the pricing problem. All algorithms are implemented in C++. Since the algorithms find a packing for a fixed bin size, we need to run the algorithms separately for every bin type.

4.2. Solving the pricing problem

The *pricing problem* is the problem of finding a feasible packing p_k of a single bin with minimum reduced cost $c_{p_k}^{\pi}$. If we define the profit s_i of every item $i \in \mathscr{R}$ as $s_i := -\pi_i$ then the pricing problem consists of m 2D knapsack problems (2DKP) defined on profits s_i, item sizes w_i and h_i and the knapsack size W_k and H_k. The best solution of the 2DKP among the m bins is a solution to our pricing problem. Recent algorithms for the 2DKP include Caprara and Monaci [8], Fekete and Schepers [14] and Pisinger and Sigurd [30]. We have chosen to use the latter framework for solving the pricing problem of the 2DVSBPP since the

solution approach is very flexible for handling additional constraints, like: guillotine cutting constraints, fixed positions, border position constraints, relative position constraints, irregularities in the bins, etc.

4.2.1. Using constraint satisfaction and branch-and-cut

Given a bin type $k \in \{1, \ldots, m\}$, profits s_i, and item sizes $w_i \times h_i$, the 2DKP can be decomposed into a 1D knapsack problem (1DKP) and a 2D constraint satisfaction problem in the following way:

1. Solve the 1DKP

$$\begin{aligned} &\text{maximize} && \sum_{i=1}^{n} s_i x_i \\ &\text{subject to} && \sum_{i=1}^{n} w_i h_i x_i \leqslant W_k H_k, \\ & && x_i \in \{0, 1\}, \qquad i = 1, \ldots, n, \end{aligned} \tag{15}$$

   using a generalization of the EXPKNAP branch-and-bound algorithm by Pisinger [29].
2. Since the 1DKP is a relaxation of the 2DKP, the solution $z = \sum_{i=1}^{n} s_i x_i$ is an upper bound on the solution of the 2DKP. Now, solve a 2D constraint satisfaction problem for the solution x of the 1DKPP to determine if items $I = \{i | x_i = 1\}$ can be packed in a bin of type k.

   (a) If a feasible packing of I is found, then this is a solution to the 2DKP.
   (b) Otherwise, we determine a minimum subset of items $I' \subseteq I$, which do not fit in a bin of type k. We add the following valid inequality to the 1DKP in step 1:

$$\sum_{i \in I'} x_i \leqslant |I'| - 1 \tag{16}$$

   This cardinality inequality eliminates the solution x from the 1DKP in step 1.

   We repeat the process of solving the 1DKP and the constraint satisfaction problem until either the solution of the 1DKP has nonnegative reduced cost or x is a feasible packing.

The inequalities found in step 2b are globally valid, which means that all the inequalities are available for the solution of the pricing problem in the following iterations—also when branching has occurred. However, in general, every inequality is only valid for the bin type for which the 2D constraint satisfaction problem was solved and for smaller bins, i.e. bins where both the widths and heights are smaller than the width and height of the current bin.

4.2.2. A heuristic pricing algorithm

The 2DKP described in the previous section is $\mathcal{NP}$-hard, hence to speed up the column generation the pricing problem is solved heuristically whenever possible. We use a heuristic algorithm in every iteration of the column generation procedure, and only when it fails to find a single bin packing with negative reduced cost, we apply the exact 2D knapsack algorithm from Section 4.2.1.

The heuristic pricing algorithm is identical to the greedy algorithm in Pisinger and Sigurd [30]. It starts out with an empty bin of type k and greedily places items in the bin until no more items can be placed. Two greedy strategies for choosing the next item to place and two greedy strategies for choosing the place to put the item have been implemented. The two strategies for choosing an item also exist in randomized version, where the probability for choosing an particular item is proportional to its greedy score. See Pisinger and Sigurd [30] for more details.

Combining the strategies for choosing an item and a free space we obtain eight variants of the heuristic pricing algorithm. In every step of the column generation procedure, we successively run one of the eight variants until a bin packing with negative reduced cost is found, or 50 runs of the randomized heuristic have been performed.

4.3. The primal bound

The column generation procedure provides fairly tight lower bounds for the optimal solution to the 2DVSBPP. To bound the optimal solution from above we start out by running a heuristic algorithm which finds a feasible solution, feeding the column-generation with a set of columns corresponding to this solution. For this purpose a simple heuristic was constructed based on a greedy approach. The algorithm repeatedly solves a maximum profit 2DKP until all items have been packed. Initially the profit of an item is set to the area. In the following iterations the items in the last bin get their profit increased by 10% to make them

Item type 1	w_r uniformly random in $[\frac{2}{3}W, W]$, h_r uniformly random in $[1, \frac{1}{2}H]$
Item type 2	w_r uniformly random in $[1, \frac{1}{2}W]$, h_r uniformly random in $[\frac{2}{3}H, H]$
Item type 3	w_r uniformly random in $[\frac{1}{2}W, W]$, h_r uniformly random in $[\frac{1}{2}H, H]$
Item type 4	w_r uniformly random in $[1, \frac{1}{2}W]$, h_r uniformly random in $[1, \frac{1}{2}H]$
Class I	w_r and h_r uniformly random in $[1, 10]$, $W = H = 10$
Class II	w_r and h_r uniformly random in $[1, 10]$, $W = H = 30$
Class III	w_r and h_r uniformly random in $[1, 35]$, $W = H = 40$
Class IV	w_r and h_r uniformly random in $[1, 35]$, $W = H = 100$
Class V	w_r and h_r uniformly random in $[1, 100]$, $W = H = 100$
Class VI	w_r and h_r uniformly random in $[1, 100]$, $W = H = 300$
Class VII	item type 1 with probability 70%, type 2, 3, 4 with probability 10% each. $W = H = 100$
Class VIII	item type 2 with probability 70%, type 1, 3, 4 with probability 10% each. $W = H = 100$
Class IX	item type 3 with probability 70%, type 1, 2, 4 with probability 10% each. $W = H = 100$
Class X	item type 4 with probability 70%, type 1, 2, 3 with probability 10% each. $W = H = 100$

Fig. 1. Instance classes considered.

more attractive. The algorithm is repeated 50 times, using an upper limit of 100 000 branch-and-bound nodes for solving the 2DKP.

The heuristic is motivated by the fact that the 2DKP will prefer to use the small items in the first bins in order to get a good filling ratio. Increasing the profits of the last (large) items motivates the 2DKP algorithm to pack the difficult rectangles early in the process. A similar approach named *sequential value correction* has been used by Mukhacheva and Zalgaller [27], Belov [3], and Kupke [20].

We also improve the primal bound during the branch-and-price algorithm by solving the IP variant of the restricted master problem (13) to optimality by CPLEX every 50 iterations of the column generation. No time limit was used for solving the problem as it generally is easy to solve.

4.4. Branching

The LP-solutions to (12) found by column generation are not necessarily integral. In order to obtain optimal integral solutions we apply a branching rule which excludes fractional solutions. In Pisinger and Sigurd [30] we described a branching strategy for the 2DBPP. This strategy can be applied to the 2DVSBPP as well.

We use the following branching rule: choose two items i_j and i_l, $j \neq l$. We divide the solution space into two subsets. On one branch we demand that items i_j and i_l may not be in the same bin. On the other branch we demand that i_j and i_l have to be in the same bin. This branching scheme changes the pricing problem, since we must disallow packings violating the branching constraints enforced. This can be handled effectively as described in [30].

5. Computational results

We have implemented the lower bounds presented in Section 3 and a branch-and-price algorithm for solving 2DVSBPP as described in the previous sections. For this purpose we used ABACUS ("A Branch-And-CUt System") [31] which is a collection of C++ classes that significantly reduces the work of implementing branch-and-bound-like algorithms. ABACUS provides an interface to CPLEX 7.0 [19] which we have used to solve the linear programs resulting from the column generation. CPLEX 7.0 has also been used to solve the integer programs of the IP area covering lower bound of Section 3.1. All tests have been carried out on an Intel Pentium III-933 with 1 GB of memory.

5.1. Test instances for the 2DVSBPP

To the best of our knowledge the only instances published (see OR-library, [2]) for the 2DVSBPP are those by Hopper and Turton [18]. However, these instances have an upper bound on the use of each bin type, and the bin costs are uniform. Hence, we propose a new set of test instances based on generalizations of the normal 2DBPP. The characteristics of the instances proposed by Berkey and Wang [5] and Lodi et al. [21] are summarized in Fig. 1.

Based on these test instances we have created a new set of instances with 5 bin types in each instance. The bin sizes have been picked uniformly at random from $[W/2, W] \times [H/2, H]$, where W, H are the bin width and height from the original problem.

The costs c_k are a function of the bin areas $W_k H_k$ as follows:

$$c_k = \left\lceil 10 \cdot \frac{1.2 \cdot W_k H_k - 0.2 \cdot WH}{WH} \right\rceil.$$

This cost function impose a relatively cheaper cost on smaller bins. For every problem class, we have created 10 problems with 20, 40, 60, 80 and 100 items.

5.2. Results

In Table 1, we have shown the results of our exact algorithm on the above test instances. Each row in the table is an average of 10 instances. Columns 3–6 show the average percentage gap between the lower bounds L_{2w}, L'_{2c}, L_{IP}, L_{DW} and the best known upper bound U. The gaps are calculated as $(U - L)/U$. Column 7 reports how many instances of 10 that were solved to optimality within the time limit of 3600 s. Column 8 reports the overall CPU-time used, while column 9 reports the average number of pricing problems solved in (13). A zero in this column means that the algorithm failed to solve the first pricing problem. Column 10 reports the number of cuts (16) added to the knapsack problem (15), while column 11 reports the number of nodes in the main branch-and-price algorithm.

In Tables 2 and 3 we have summarized the results grouped by test instance classes and instance sizes.

The results show that the 2DVSBPP in general is harder to solve than the normal 2DBPP. We are not able to solve as many 2DVSBPP instances to optimality as was the case for the 2DBPP in Pisinger and Sigurd [30]. As expected, Table 3 shows us that the larger problems are much harder to solve than the smaller ones. However, there is a big difference in difficulty between the instance classes as seen in Table 2. We are able to solve all instances from class *IX* to optimality, since this class have many large items, so each bin will only contain a few items. This means that the number of different packings of a single bin is relatively small, which implies that our master problem will also be quite small, making it easier to solve to IP optimality. In the other end of the scale, we are able to solve very few of the instances in classes *II*, *IV* and *VI* which all contain many small items.

The quality of the lower bounds, in terms of the average gap between the lower bound and the best known solution value, appears to be independent of the instance sizes. But, there is some variation in the quality of the bounds between the instance classes.

In general, the L_{2w} lower bound performs bad, which could be expected, since it only considers the most expensive item to cover, disregarding that we need to cover *all* items. The L'_{2c} and L_{IP} lower bound performs much better, L_{IP} being only slightly better than L'_{2c}. These lower bounds could possibly be useful in a branch-and-bound algorithm for solving the exact 2DVSBPP. Since L'_{2c} can be computed in $O(nm)$ time whereas finding L_{IP} is $\mathcal{NP}$-hard, using L'_{2c} seems most promising of the two. However, the lower bound obtained by Dantzig–Wolfe decomposition and LP-relaxation performs better. In the root node, this lower bound is 3.89% on average from the best known optimal solution. Since, we have only proven optimality for about one-third of the instances, the quality of the lower bound may be even better.

6. Conclusions

We have introduced the variable sized 2DBPP with variable costs. The problem is a natural generalization of the variable sized 1DBPP and the normal 2DBPP. We have presented the first lower bounds and the first exact algorithm for the problem, along with a set of test instances to evaluate lower bounds and algorithms.

Our results show that the 2DVSBPP is hard to solve even for small instances. As shown, the polynomial time computable bounds from 1DVSBPP and normal 2DBPP do not generalize well to the 2DVSBPP. In lack of good polynomial bounds, the Dantzig–Wolfe decomposition in connection with delayed column generation has proved to be a strong tool for deriving lower bounds of good quality. The branch-and-price algorithm, using this bound, has been able to solve a few instances with up to 100 items.

Since the pricing algorithm is based on constraint programming, the algorithm is easily extended to handle various additional constraints as discussed in Pisinger and Sigurd [30]. This includes guillotine cutting constraints, various requirements to the relative position of the items, irregularities in the bins, etc. As the addition of further constraints decreases the size of the solution space, we can hope that the pricing problem will be easier to solve. The lower bounds presented in Section 3 are all valid for the extended problems, although it is not obvious how to modify the bounds to make use of the additional constraints.

In order to solve larger instances of the 2DVSBPP a number of directions can be followed. The pricing algorithm is called numerous times in the branch-and-price algorithm, and hence any improvement in time complexity of this algorithm is of great importance. Tighter, polynomial bounds could be used in a pure branch-and-bound algorithm. Also, reduction techniques which a-priori may fix some decision variables at their optimal value, could be considered.

Table 1
Results for problem classes I–X

Class	Items	Avg. gap L_{2w} (%)	Avg. gap L'_{2c} (%)	Avg. gap L_{IP} (%)	Avg. gap L_{DW} (%)	# Solved optimally	Avg. CPU (seconds)	Avg. cols. generated	Avg. cuts added	Avg. B&B nodes
I	20	84.00	12.80	11.52	0.48	10	21	206	256	75
I	40	91.54	7.78	6.85	0.51	7	1196	5658	3243	5951
I	60	94.51	8.18	7.69	0.60	1	3256	3717	12 838	7260
I	80	96.03	7.02	6.66	0.52	4	2679	1344	16 147	2378
I	100	96.70	7.40	7.03	1.42	2	3267	640	21 368	821
II	20	49.32	5.48	5.48	5.48	1	3242	498	66	3
II	40	72.00	12.80	8.80	3.20	1	3246	17 254	61	2
II	60	81.25	10.80	6.25	6.82	0	3600	8751	1849	2
II	80	87.34	8.86	5.91	8.02	2	2934	2899	909	3
II	100	88.61	12.03	9.81	11.39	0	3600	4619	642	3
III	20	77.80	16.95	14.56	0.24	10	26	99	310	32
III	40	87.91	15.87	14.29	1.10	5	1862	1343	2957	1131
III	60	92.35	14.06	13.37	1.47	2	3265	1013	13 540	1542
III	80	94.75	14.65	14.09	2.63	3	2982	662	18 431	34
III	100	95.30	12.54	12.31	3.10	2	3086	943	23 653	5
IV	20	47.14	18.57	17.14	14.29	2	2883	877	57	3
IV	40	71.21	18.18	14.39	6.06	0	3600	16 035	1283	2
IV	60	82.39	11.93	9.66	7.95	1	3290	8045	262	2
IV	80	86.11	11.90	10.32	9.52	0	3600	3541	585	3
IV	100	86.14	18.37	15.36	15.36	0	3600	4716	306	3
V	20	82.30	19.47	17.52	0.71	10	16	82	241	37
V	40	90.79	15.10	14.27	0.64	6	1539	3698	2194	3855
V	60	93.99	13.65	12.99	0.72	4	2353	760	11 745	834
V	80	95.61	12.47	12.29	0.83	4	2473	602	17 665	161
V	100	96.36	13.12	12.76	2.59	0	3600	827	27 773	68
VI	20	48.44	18.75	15.63	12.50	3	2525	211	29	2
VI	40	70.00	17.27	10.91	5.45	0	3600	17 059	72	2
VI	60	79.75	13.92	8.23	6.33	0	3600	10 368	80	2
VI	80	84.96	16.81	14.60	11.95	0	3600	0	6	1
VI	100	87.73	16.25	14.44	16.97	0	3600	0	5	1

VII	20	79.84	17.28	14.81	2.47	9	512	8824	796	8517
VII	40	90.07	12.71	12.12	1.89	1	3473	2894	11 811	2668
VII	60	93.03	10.39	9.69	1.88	0	3600	347	19 779	3
VII	80	95.25	13.30	12.73	1.99	0	3600	410	23 490	3
VII	100	96.05	11.02	10.75	2.21	0	3600	707	28 261	3
VIII	20	80.47	17.16	14.60	1.97	9	807	12 888	938	6734
VIII	40	90.08	13.39	12.60	1.19	6	2313	2048	7984	2056
VIII	60	93.26	12.33	11.66	1.82	0	3600	290	20 633	5
VIII	80	95.20	11.94	11.51	2.35	0	3600	478	21 262	3
VIII	100	96.09	11.51	11.08	2.00	0	3600	558	27 543	3
IX	20	91.42	25.56	24.44	0.00	10	2	7	294	1
IX	40	95.69	25.85	25.55	0.04	10	20	12	894	1
IX	60	97.24	27.17	27.01	0.00	10	59	14	2156	1
IX	80	97.91	25.06	24.87	0.00	10	177	24	3562	2
IX	100	98.24	23.75	23.57	0.00	10	369	52	5680	2
X	20	74.93	17.41	13.72	1.58	7	1112	121	336	8
X	40	85.32	13.07	11.60	1.91	2	2935	513	5919	40
X	60	89.65	12.22	11.59	4.24	0	3600	947	22 522	8
X	80	91.82	13.18	12.36	4.01	0	3600	1082	27 328	3
X	100	93.46	11.18	10.52	4.05	0	3600	1803	29 607	3
Avg.		86.35	14.57	13.04	3.89	164	2526.40	3009.72	8787.36	885.64

Table 2
Summarized results grouped by test instance classes

Class	*Avg. gap* L_{2w} (%)	Avg. gap L'_{2c} (%)	Avg. gap L_{IP} (%)	Avg. gap L_{DW} (%)	# Solved optimally	Avg. CPU (seconds)	Avg. cols. generated	Avg. cuts added	Avg. B&B nodes
I	94.55	7.87	7.34	0.83	24	2083.8	2313	10 770.4	3297
II	81.55	10.57	7.66	8.09	4	3324.4	6804.2	705.4	2.6
III	92.48	14.14	13.41	2.20	22	2244.2	812	11 778.2	548.8
IV	80.56	15.49	12.99	11.12	3	3394.6	6642.8	498.6	2.6
V	94.00	13.74	13.20	1.36	24	1996.2	1193.8	11 923.6	991
VI	80.12	16.29	12.93	11.74	3	3385	5527.6	38.4	1.6
VII	93.42	12.16	11.54	2.06	10	2957	2636.4	16 827.4	2238.8
VIII	93.47	12.41	11.74	1.95	15	2784	3252.4	15 672	1760.2
IX	97.15	25.21	24.95	0.01	50	125.4	21.8	2517.2	1.4
X	89.64	12.66	11.62	3.58	9	2969.4	893.2	17 142.4	12.4

Table 3
Summarized results grouped by test instance sizes

Items	Avg. gap L_{2w} (%)	Avg. gap L'_{2c} (%)	Avg. gap L_{IP} (%)	Avg. gap L_{DW} (%)	# Solved optimally	Avg. CPU (seconds)	Avg. cols. generated	Avg. cuts added	Avg. B&B nodes
20	81.81	18.95	17.02	1.33	71	1114.6	2381.3	332.3	1541.2
40	90.50	16.49	15.54	1.00	38	2378.4	6651.4	3641.8	1570.8
60	93.79	16.06	15.41	1.35	18	3022.3	3425.2	10540.4	965.9
80	95.47	15.43	14.98	1.65	23	2924.5	1104.2	12938.5	259.1
100	96.14	14.62	14.21	2.29	14	3192.2	1486.5	16483.8	91.2

References

[1] C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, P.H. Vance, Branch-and-price: column generation for solving huge integer programs, Oper. Res. 46 (1998) 316–329.
[2] J.E. Beasley, OR-library, 2004, http://mscmga.ms.ic.ac.uk/info.html.
[3] G. Belov, Problems, models and algorithms in one- and two-dimensional cutting, Ph.D. Thesis, Technischen Universität Dresden, 2003, http://www.math.tu-dresden.de/∼belov/publ/text030908_SUBMIT.pdf.
[4] G. Belov, G. Scheithauer, A cutting plane algorithm for the one-dimensional cutting stock problem with multiple stock lengths, European J. Oper. Res. 141 (2002) 274–294.
[5] J.O. Berkey, P.Y. Wang, Two dimensional finite bin packing algorithms, J. Oper. Res. Soc. 38 (1987) 423–429.
[6] M.A. Boschetti, A. Mingozzi, The two-dimensional finite bin packing problem, Part I: new lower bounds for the oriented case, 4OR 1 (2003) 27–42.
[7] M.A. Boschetti, A. Mingozzi, The two-dimensional finite bin packing problem, Part II: new lower and upper bounds, 4OR 2 (2003) 135–148.
[8] A. Caprara, M. Monaci, On the two-dimensional knapsack problem, Oper. Res. Lett. 32 (2004) 5–14.
[9] C.S. Chen, S.M. Lee, Q.S. Shen, An analytical model for the container loading problem, European J. Oper. Res. 80 (1995) 68–76.
[10] E.G. Coffmann, M.R. Garey, D.S. Johnson, Approximation algorithms for bin packing: a survey, in: D.S. Hochbaum (Ed.), Approximation Algorithms for NP-hard Problems, PWS Publishing, Boston, 1997, pp. 46–93.
[11] J. Csirik, G.J. Woeginger, On-line packing and covering problems, in: A. Fiat, G.J. Woeginger (Eds.), Online Algorithms, Springer, Berlin, 1998, pp. 147–177.
[12] M. Dell'Amico, S. Martello, D. Vigo, A lower bound for the non-oriented two-dimensional bin packing problem, Discrete Appl. Math. 118 (2002) 13–24.
[13] S.P. Fekete, J. Schepers, New classes of lower bounds for the bin packing problem, Math. Programming 91 (2001) 11–31.
[14] S.P. Fekete, J. Schepers, An exact algorithm for higher-dimensional orthogonal packing. Oper. Res., to appear.
[15] D.K. Friesen, M.A. Langston, Variable sized bin packing, SIAM J. Comput. 15 (1) (1986) 222–230.
[16] P.C. Gilmore, R.E. Gomory, A linear programming approach to the cutting stock problem, Oper. Res. 9 (1961) 849–859.
[17] P.C. Gilmore, R.E. Gomory, A linear programming approach to the cutting stock problem—part II, Oper. Res. 13 (1963) 94–119.
[18] E. Hopper, C.H. Turton, An empirical study of meta-heuristics applied to 2D rectangular packing problem, Studia Inform. 2 (2002).
[19] ILOG, CPLEX 7.0, Reference Manual, ILOG, S.A., France, 2000.
[20] J. Kupke, Lösung von ganzzahligen verschnittproblemen mit branch-and-price, Diplomarbeit, Universität zu Köln, 1998.
[21] A. Lodi, S. Martello, D. Vigo, Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems, INFORMS J. Comput. 11 (1999) 345–357.
[22] A. Lodi, S. Martello, D. Vigo, Recent advances on two-dimensional bin packing problems, Discrete Appl. Math. 123 (2002) 379–396.
[23] S. Martello, D. Pisinger, D. Vigo, The three-dimensional bin packing problem, Oper. Res. 48 (2000) 256–267.
[24] S. Martello, P. Toth, Knapsack Problems: Algorithms and Computer Implementations, Wiley, Chichester, 1990.
[25] S. Martello, D. Vigo, Exact solution of the two-dimensional finite bin packing problem, Management Sci. 44 (1998) 388–399.
[26] M. Monaci, Algorithms for packing and scheduling problems, Ph.D. Thesis, University of Bologna, 2002.
[27] E.A. Mukhacheva, V.A. Zalgaller, Linear programming for cutting problems, Internat. J. Software Eng. Knowledge Eng. 3 (1993) 463–477.
[28] H. Onodera, Y. Taniguchi, K. Tmaru, Branch-and-bound placement for building block layout, 28th ACM/IEEE Design Automation Conference, 1991, pp. 433–439.
[29] D. Pisinger, An expanding-core algorithm for the exact 0-1 knapsack problem, European J. Oper. Res. 87 (1995) 175–187.
[30] D. Pisinger, M.M. Sigurd, Using decomposition techniques and constraint programming for solving the two-dimensional bin packing problem, Technical Report DIKU-03/01, Department of Computer Science, University of Copenhagen, Denmark, 2002.
[31] S. Thienel, ABACUS—A Branch-And-CUt System, Ph.D. Thesis, Universität zu Köln, 1995.
[32] H.P. Williams, Model Building in Mathematical Programming, fourth ed., Wiley, Chichester, 1999.